## Análisis de capacidad

### Pruebas:

Las pruebas realizadas para esta iteración fueron tomadas y extendidas de la iteración anterior. En este caso utilizamos 8 hilos de ejecución cada uno realizando 5 pruebas en un tiempo distribuido entre 15 a 40 segundos. Para ello utilizamos el software para pruebas de rendimiento JMeter que hacía peticiones al proyecto alojado en la nube AWS.

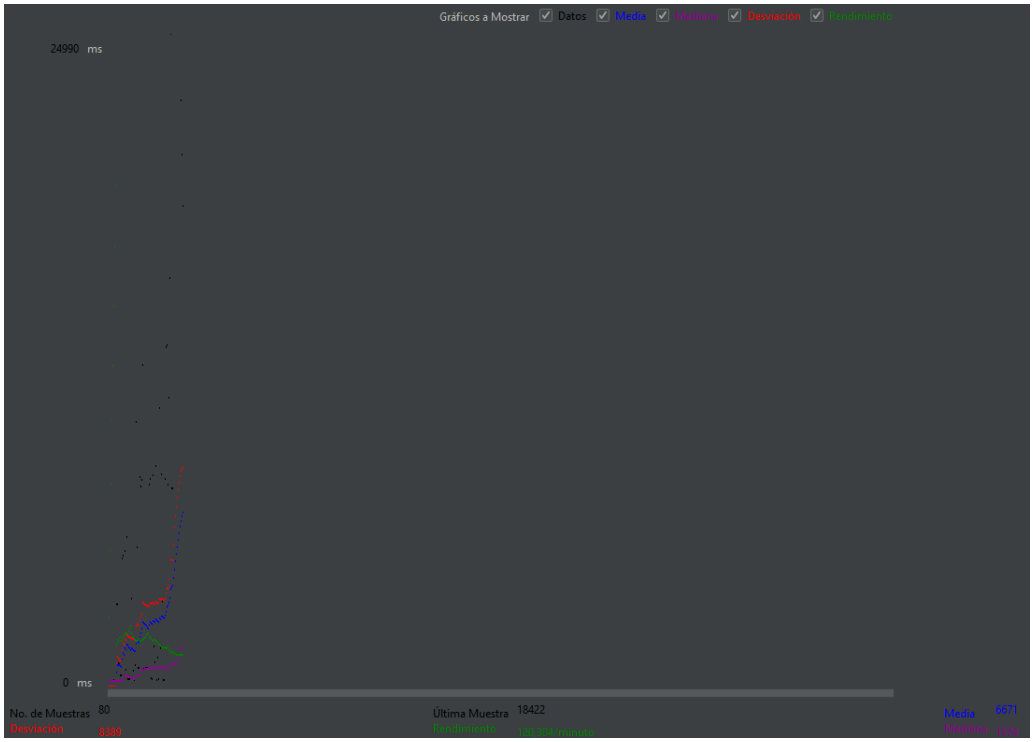
Los casos por evaluar en este análisis de capacidad son:

- Caso 1: Inicialmente se utiliza una sola instancia y se hacen peticiones a la API a través del balanceador de carga. En dado caso de que la máquina virtual donde esta alojada la API llega a un consumo de CPU del 40%, se escala horizontalmente para que el balanceador de carga distribuya las peticiones en 2 máquinas virtuales. En el caso del worker, al llegar la cola a 10 peticiones se escala horizontalmente para distribuir las peticiones, Esto hasta llegar a un máximo de 3 máquinas virtuales con el worker.

- Caso 2: No se utilizan máquinas virtuales sino el servicio de GCP Google Cloud Run. Esta es una herramienta que permite ejecutar contenedores de aplicaciones de forma escalable. Para este caso Cloud Run no se configura para tener características autoescalables.

### Caso 1:

En este caso se aprecia como la curva de rendimiento se estabiliza en las primeras peticiones en un nivel cercano a 120,304/minuto y aumenta probablemente al momento en el que escala horizontalmente el sistema.



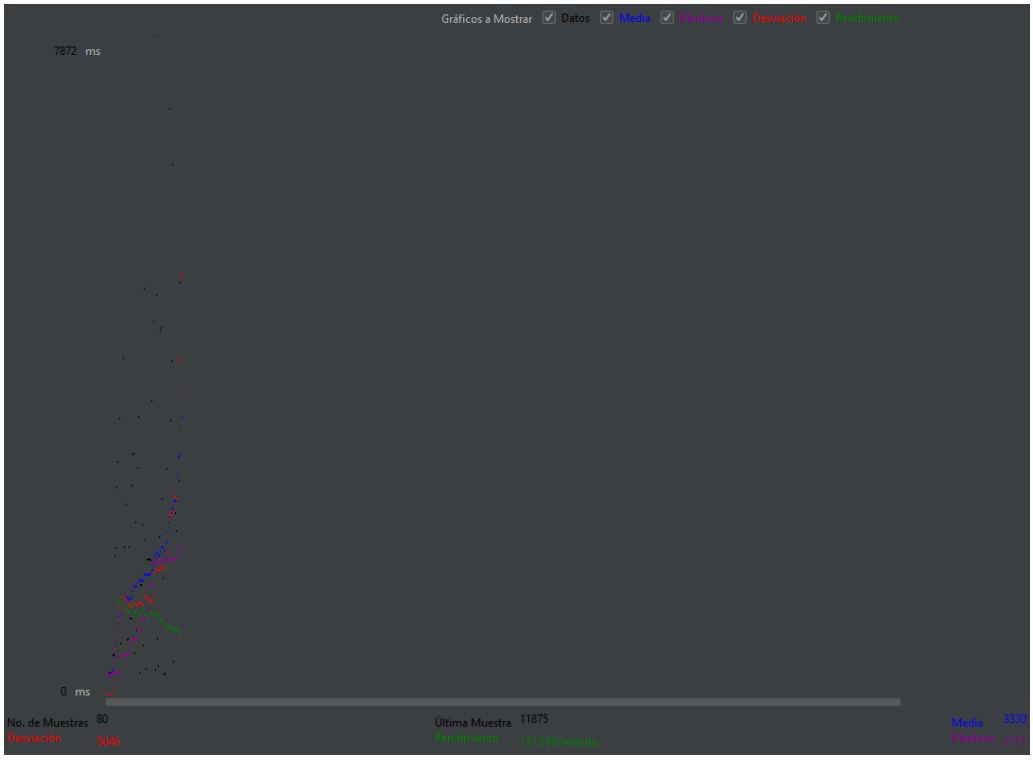

Observando la tabla de resultados, se evidencia una ejecución correcta de todas las pruebas e hilos. Un rendimiento ligeramente mejor de las API que crea la tarea que es encolada y eventualmente procesada por el worker. Respecto al Worker, en este caso no se tiene un mejor rendimiento, pero dependiendo del sistema en cuestión o los requisitos de calidad puede ser un valor aceptable.

| Etiqueta | # Muestras | Media | Min | Máx | Desv. Estándar | % Error | Rendimiento | Kb/sec | Sent KB/sec | Media de Bytes |
|---|---|---|---|---|---|---|---|---|---|---|
| Login | 40 | 542 | 230 | 1574 | 337,34 | 0,00% | 1,6/sec | 0,78 | 0,44 | 486,8 |
| Create_task | 40 | 12799 | 3161 | 27617 | 8095,00 | 0,00% | 1,0/sec | 0,25 | 2954,48 | 249,4 |
| Total | 80 | 6671 | 230 | 27617 | 8389,41 | 0,00% | 2,0/sec | 0,72 | 2937,57 | 368,1 |

| Etiqueta | # Muestras | Media | Min | Max | Desv. Estandar | % Error | Rendimiento (s) | Kb/sec | Sent KB/sec | Media Bytes |
|---|---|---|---|---|---|---|---|---|---|---|
| Login | 40 | 542 | 230 | 1574 | 337.34 | 0 | 1.6 | 0.78 | 0.44 | 486.8 |
| Create_task | 40 | 12799 | 3161 | 27617 | 8095 | 0 | 1 | 0.24 | 2954.48 | 249.4 |
| Total | 80 | 6671 | 230 | 27617 | 8389.41 | 0 | 2 | 0.72 | 2937.57 | 368.1 |

Caso 2:

En este caso de estudio se puede apreciar una dispersión inicial menor que en otras iteraciones. El rendimiento es mucho mayor estabilizándose en un valor de 137,559/minuto dando una idea de las eficiencias que se pueden llegar a tener con los servicios Pass de GCP.




Viendo más a detalle el comportamiento de los request desarrollados en esta prueba, hay una media en los tiempos de respuesta significativamente menor, pero rendimientos ligeramente menores evidenciado en los request generados a la API de login.

| Etiqueta | # Muestras | Media | Min | Máx | Desv. Estándar | % Error | Rendimiento | Kb/sec | Sent KB/sec | Media de Bytes |
|---|---|---|---|---|---|---|---|---|---|---|
| Login | 40 | 1059 | 249 | 7011 | 1399,02 | 0,00% | 1,5/sec | 0,82 | 0,40 | 547,0 |
| Create_task | 40 | 5602 | 1584 | 29928 | 6216,94 | 0,00% | 1,2/sec | 0,35 | 3383,53 | 311,0 |
| Total | 80 | 3330 | 249 | 29928 | 5046,27 | 0,00% | 2,3/sec | 0,96 | 3358,92 | 429,0 |

| Etiqueta | # Muestras | Media | Min | Max | Desv. Estandar | % Error | Rendimiento (s) | Kb/sec | Sent KB/sec | Media Bytes |
|---|---|---|---|---|---|---|---|---|---|---|
| Login | 40 | 1059 | 249 | 7011 | 1399.02 | 0 | 1.5 | 0.82 | 0.40 | 547 |
| Create_task | 40 | 5602 | 1584 | 29928 | 6216.94 | 0 | 1.2 | 0.35 | 3383.53 | 311 |
| Total | 80 | 3330 | 249 | 29928 | 5046.27 | 0 | 2.3 | 0.96 | 3358.92 | 429 |

## Análisis de resultados del sistema:

En este apartado se van a evaluar los resultados obtenidos por la herramienta Monitoring de GCP. Se tomaron los resultados del caso 2 de la iteración pasada, donde se evaluó el sistema utilizando herramientas IaaS con auto escalamiento. Y se tomaron los resultados obtenidos de la misma herramienta para esta iteración donde se utiliza Cloud Run que ejecuta un container con la app.

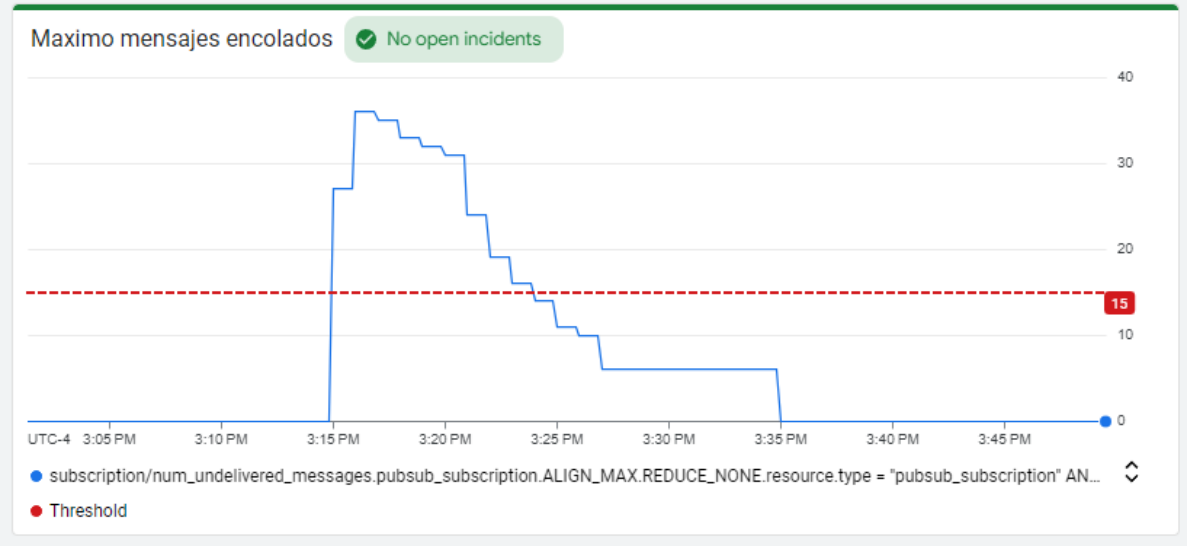
Se debe tener en cuenta que para el caso 2, al estar toda la aplicación en un container no se tiene un análisis tan detallado del web server y del worker pues la herramienta configurada permite evaluar los datos del servicio más no del container y el programa

que ejecuta. Para este caso se evaluará entonces la velocidad con la que se desencolan los mensajes y el consumo de CPU.

Análisis mensajes encolados:

Caso 1:

La velocidad del sistema a pesar de tener la habilidad de autoescalar en comparación con el caso 2, el procesamiento de videos no es tan rápido y se puede evidenciar en el pico máximo de 36 tareas encoladas. El tiempo para procesa las 40 tareas entonces es de poco menos de 20 minutos.



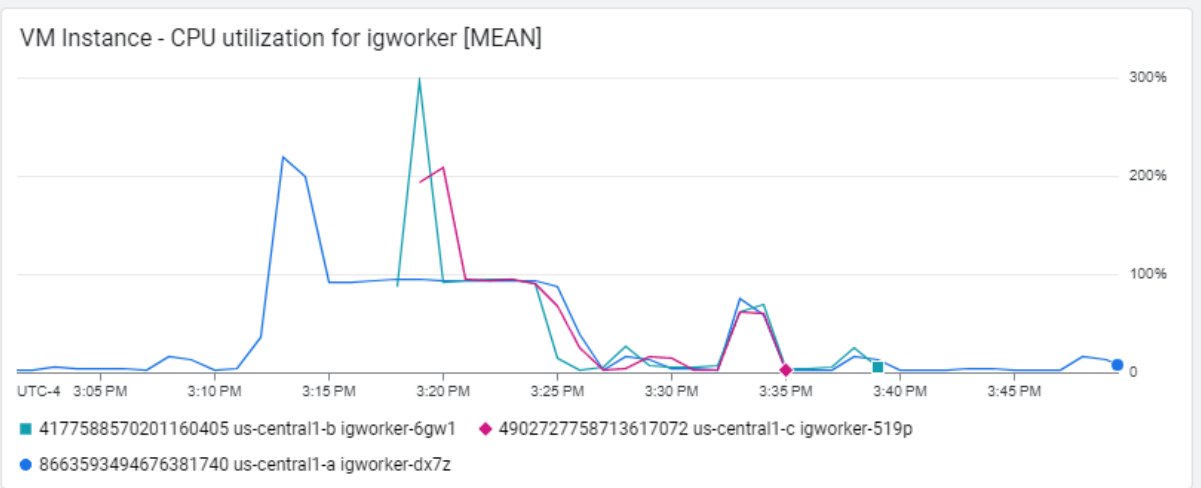

Caso 2:

Para este caso, se puede ver lo eficiente que es el proceso utilizando Cloud Run. En total procesar las 40 tareas toma aproximadamente 4 minutos con un pico máximo de tareas encoladas de 24.

Uso de CPU

Caso 1:

Al ser el caso 1 un proceso con la habilidad de autoescalar, en estas pruebas se generaron hasta 3 máquinas virtuales para atender la carga del procesamiento de los 40 videos. El sistema muestra una especie de sobre carga al iniciar la máquina virtual para eventualmente consumir casi todos los recursos disponibles en las 3 máquinas virtuales. Como se menciona anteriormente, es un proceso de aproximadamente 20 minutos donde es válido tener en cuenta los gastos generados por el uso de 3 máquinas virtuales.




Caso 2:

Para el segundo caso, el servicio de Cloud Run que ejecutaba el container con la aplicación no tenía la habilidad de autoescalar, sin embargo, se tarda aproximadamente 4 minutos con un pico de consumo de CPU de hasta 63 %. en comparación con el caso anterior, un menor tiempo y un consumo menor de CPU en un solo servicio. Esto en la operación normal de una solución de software pueden ser costos asociados importante a tener en cuenta.